ICASE

RASTER SCAN CONVERSION USING CONCURRENT MICROPROCESSORS

Griffith Hamlin, Jr.

Report Number 77-9

May 10, 1977

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

RASTER SCAN CONVERSION
USING
CONCURRENT MICROPROCESSORS

Griffith Hamlin Jr.

ABSTRACT

Raster scan computer graphics displays require the image generated by the program to be converted to raster scan order. This is almost always done using a large frame buffer memory. An alternative technique is to substitute for the frame buffer memory enough processing power to perform the conversion "on the fly" for each frame. It appears that microprocessors can now provide this processing power at low cost. One possible implementation of such a raster-scan conversion algorithm is presented which uses one LSI microprocessor and one small special purpose processor running concurrently in a pipelined fashion. With today's microprocessor technology, this approach is shown to be feasible and its economics compare favorably with a frame buffer system of similar performance.

This report will be presented at, and will appear in the proceedings of, The First Annual Rocky Mountain Symposium on Microcomputers: Systems Software Architecture. It was prepared as a result of work performed under NASA Contract No. NAS1-14101 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

RASTER-SCAN CONVERSION USING CONCURRENT MICROPROCESSORS

I. INTRODUCTION

Computer graphics display devices can be classified as either random or raster scan devices. Random scan devices allow the image to be drawn on the display in any order generated by an application program. For line drawings, this is often specified by a list of vector endpoints. Raster scan devices are constrained to display the image according to some specific order, usually left to right along horizontal scanlines. Therefore, before displaying an image generated by a program, the data must somehow be sorted so that it is available to the raster display device in the proper order. This raster scan conversion process poses an added complexity for raster scan displays. However, once this conversion is accomplished, raster scan displays have some advantages over random scan.

First, constraining the deflection of a CRT display to follow a fixed raster-scan pattern considerably simplifies the analog deflection electronics needed. Ordinary television receivers are raster scan CRT devices. Compared to available random scan computer display devices they are inexpensive, provide color, and require much less adjustment of the analog deflection circuitry. Also they are widely used.

Second, the time required to display one frame with raster scan devices is a constant (1/30 sec usually). Random scan devices, however, usually require a display time roughly proportional to the total length of all vectors being displayed. Complex pictures may require too much time to display (over 1/30 sec) and therefore appear to flicker. On raster scan devices an arbitrarily complex image can be displayed without flicker provided it is specified within the resolution limits of the raster scan display. Thus raster scan devices are usually used when it is desirable to display surfaces, which require considerably more displayed vector length than corresponding line drawings. However, raster scan devices suffer from a "stair stepping" effect when used to draw non-horizontal or non-vertical lines. Random scan devices do not suffer from this effect.

II. APPROACHES TO RASTER-SCAN CONVERSION

A frame buffer memory is almost universally used to accomplish the raster-scan conversion process. buffer memory. One word of this memory is assigned to each resolvable (x,y) position on the display so that increasing addresses scan the display screen in the raster-scan order. The contents of any one word of this memory specify the intensity/color of the associated position on the screen. The frame buffer memory is loaded in any order required by the program with intensity/color information describing an image. The raster-scan output is then produced by scanning the memory sequentially from its lowest to highest address. The operation performed here is actually a "bucket sort" with each word of the frame buffer being one bucket capable of holding one datum.

Using a frame buffer memory with high resolution or many intensity/color levels requires much memory. For example, a 512x512 resolution with 512 intensity/color combinations requires 512x512x9 = 2,359,296 bits of memory. Also, to provide real time motion, each consecutive frame may display different images. For this, the memory speed must be fast enough to allow a new image to be written into the frame buffer within one frame time (1/30 sec). This writing time is a function of the complexity of the picture. In general, writing must proceed one word at a time since access is random. Of course, if real-time motion is not desired, the frame buffer can be filled slowly, after which it can be displayed for many frames.

An alternative approach is to perform the raster conversion process by sort techniques that do not require a large memory. With this approach, enough processing power is required for the entire sort to be done "on the fly" for each frame, even for systems without real-time motion. This approach therefore would seem to be useful for systems which need real time motion. Also, the complexity of a moving image that can be handled in real time is now determined by the speed of this processing rather than by the speed of the frame buffer memory. Using the approach described below, the speed of the processors must grow linearly with resolution to display an image of a given complexity. Frame buffer memory size and speed grow as the square of the resolution. Thus the processor sort approach appears useful for systems requiring high resolution. Jordan and Barrett[1] proposed one such conversion algorithm for line drawings. Earlier, Erdahl[2] described the design of hardware for executing the last portion of the scan conversion process for surface drawings. More recently, Meyer[3] has reported on a system in operation with hardware for this purpose.

This hardware, made by Staudhammer and associates[4], generates video in real time from run length encodings of the images.

Large frame buffers for high resolution diplays are becomming economically feasible due to the dropping cost of memory for frame buffers. However, processor cost is also dropping with the advent of inexpensive microprocessors. In the following section, a method of raster-scan conversion is presented which could be implemented using several small processors running concurrently. Such a system would be capable of moderate resolution and real time motion of moderately complex images. It is argued that the processors now becomming readily available have the capability of performing the raster scan conversion process and that because of their cost relative to memory costs, this approach currently compares favorably with a frame buffer system of similar parameters.

III. RASTER-SCAN CONVERSION PROCEDURE

In order to show the feasibility of using concurrent processors to implement the approach described in this section, we will assume reasonable resolution and picture complexity parameters, determine the required processing speeds, and then present one possible design using these processors that could be implemented from readily available microprocessor components. Finally we make a comparison of hardware requirements of this implementation with the requirements of a frame buffer implementation. Specifically,

1. Assume 512x512 resolution. This is adequate for many purposes.

2. Assume 9 bits of intensity/color levels (say 8 levels of each of 3 colors).

3. Assume the picture complexity is at most 2000 straight vectors (for line drawings) or 2000 surface edges (in the case of surface drawings). This number was obtained by counting lines on several drawings of aircraft and spacecraft obtained from engineers involved in vehicle analysis at NASA Langley Research Center.

4. Assume the maximum number of vectors (surface edges) that intersect any horizontal scan line is 500. This is 25 percent of the entire picture. The drawings mentioned in (3) above had at most 13 percent of their vectors on any one scan line.

5. Assume a refresh rate of 30 frames/second.

In order to compare the cost of implementing this raster-scan conversion method with the frame-buffer method, we must be able to compare processors with some equivalent amount of memory. A quick search through the microcomputer literature at the time of this writing reveals that a microprogrammable processor can be obtained for approximately the cost of 24K bytes of MOS memory. Such processors are today available on a few LSI circuits, are microprogrammable, can be implemented with any convenient word size (bit sliced), and can execute 5 to 10 million microinstructions per second. A processor of this class with an appropriate word size will hereafter be called a "fast microprocessor." Their cost relative to memory cost may or may not remain constant in the future. At a low level both processors and memory bits may be regarded as some number of logic gates to be fabricated onto one LSI integrated circuit. For this reason one might expect the costs of processors and memory to be at least somewhat correlated.

A. INPUT DATA AND Y-SORT PROCESSOR

The input data describe, in an encoded manner, the image to be converted to raster-scan. This consists mainly of the endpoints of vectors along with their intensity/color. To generate surface images, these vectors are taken to represent the left edge of the surface, in a manner described in section C. For this example, endpoints are given as pairs of 9 bit integers of (x,y) screen coordinates. A vector from (Xs,Ys) to (Xe,Ye) of intensity I is described by 5 9-bit words consisting of:

Xs	Ys	Xe	Ye	I

Without loss of generality, assume Ys $\leq$ Ye. We assume the existence of some computer capable of generating this list every 1/30 second if real time motion is required of the entire image. All transformations (rotation, scaling, etc.) are assumed to have been performed on this data. Alphanumeric data and other graphics commands could easily be accomodated, but are not relevant for this discussion.

The raster-scan conversion procedure described here consists of several sort and merge operations on the image data. Referring to FIGURE 1, we first sort the input data into ascending order of Ys, using a Y-sort microprocessor. This produces the Y-sorted vector list. Next, using a scan line processor, we produce a standard raster scan video signal. Each of the pipelined processors communicates with the next one by shared memory buffers. Double buffers are used so that the Y-sort processor can be processing frame n+1 while the scan line processor is processing frame n from the second buffer. For the Y-sort processor, a bucket sort would be appropriate with 512 buckets, each of variable size. This suggests a data structure consisting of a set of 512 linked lists, one list corresponding to the Y value of each scan line. Also, while sorting, the Y-sort processor should replace Xe with dx/dy = (Xs-Xe)/(Ys-Ye), calculated to 18 bits percision. A moment's reflection will show that 18 bits are needed to specify the slope with the same precision contained in the original data. To process 2000 vectors in 1/30 second requires a processor fast enough to process one vector each 33 us, on the average. This corresponds to about 200 to 300 instruction executions. A count of executed instructions in a small program written for the INTEL 3000 series[5] microprocessor shows that this "fast microprocessor" can handle the sort, slope calculation, and linked list manipulation in the required time. This microprogram performed the division in about 150 instructions, leaving 50 to 150 instructions for the rest of the processing. The memory requirements of this Y-sort are 24000 9 bit words. This provides for two buffers each capable of holding the sorted data lists. Double buffering is used so that the Y-sort processor can sort the data for frame n+1 using one buffer while the scan line processor (described below) processes data for frame n using the other buffer. Each list entry consists of five 9-bit words of data and one 9-bit link pointer to the next entry in the list. Ys need not be stored with each vector.

B. SCANLINE PROCESSOR FOR LINE DRAWINGS

As each scan line is processed, an ACTIVE LIST of all vectors intersected by the current scan line is maintained. New entries to the active list are taken from the top of the sorted vector list produced by the Y-sort processor. Vectors are deleted from the active list while processing the last scan line which intersects them. Each entry in the ACTIVE LIST consists of:

Xc	dx/dy	Ye	I

where Xc is initially set to Xs. For line drawings using this method, there is no need to sort the ACTIVE LIST. Hence the scan line processor simply processes each entry in the previous scan line's active list and then processes entries at the top of the sorted input data list (if any) that are intersected by the current scan line. The processing done to an entry from either of these two sources is the same. It consists of:

1. Calculate $Xc' = dx/dy + Xc$.
2. Place intensity/color I in a 512 word scan line buffer at all x-locations between Xc and Xc'.
3. If this vector will be intersected by the next scan line (i.e. $Ye \neq$ y-value of current scan line) place this vector into a second ACTIVE LIST buffer, replacing Xc by Xc'. Otherwise drop this vector from the ACTIVE LIST by not placing it into the second buffer. This second buffer will be used as the primary buffer on the next scan line.

The memory requirements for this process consist of two buffers to double buffer the active list each consisting of 2500 9-bit words. Also, two 512 word scan line buffers for holding the intensities (the result of this process) are needed.

The speed requirements of this processor are rather high. For a maximum size ACTIVE LIST of 500 entries, the processor must process one entry approximately every 130ns. With today's "fast microprocessor" speeds, this is an impossible situation. For realistic images, this maximum size should seldom be reached, thus relaxing the speed requirement and allowing the use of one or more "fast microprocessors" if one is willing to use a statistically average length active list and several 512 word scan line buffers to feed the video generator while processing scan lines with long active lists. This would normally have no effect on real time motion of the images. However, if all line buffers were empty when the video generator requested the next line, the display could not continue at the normal rate. An unmodified TV display will usually not operate at a reduced rate. Some type of display with a variable scan line processing rate would be well suited for this approach.

However, a relatively simple special purpose hardwired processor can perform the required function for 500 vectors in real time for each line. A design of such a unit is given in FIGURE 2. For comparison purposes, we will assume two scan line buffers and the special-purpose hardware processor of FIGURE 2. A more detailed design of this processor has been done using the readily available 7400 series logic family. It is implementable for about the hardware cost of implementing a "fast microprocessor" CPU.

To actually drive a TV or other raster device, a hardware video generator will also be needed which will accept one 512 word buffer of intensity information for each scan line and generate the video signal. A similar video generator is needed for the frame-buffer method also, so its cost will not be considered in comparing the methods.

C. SCANLINE AND ACTIVE-LIST PROCESSORS FOR SURFACES.

Shaded surface images can be processed in much the same manner as line drawings. In this case we assume that each vector in the ACTIVE LIST represents the left side of a planar polygon. This surface is assumed to extend to the right until reaching the next line to its right in the ACTIVE LIST. This requires the ACTIVE LIST to be sorted on Xc from left to right. Note that this representation of surfaces does not contain a separate right side for polygon boundaries, so overlapping surfaces cannot be represented. If the image data should contain two overlapping surfaces, one or the other of the surfaces must be displayed. If we do not display surfaces of less than one raster unit in width (below the display resolution), then the integer part of all Xc values for all vectors in the active list at any one time will be different (except for overlapping surfaces with a common left side). Thus the active list can be stored as a continuous vector in memory, using 512 consecutive addresses with each address associated with some integer value of Xc. New entries can be easily placed in the correct position in the active list. Entries may change places each scan line as Xc is updated.

The scan line processor for this scheme is similar to the processor for line drawings. A block diagram is given in FIGURE 3. This processor also merges new active list entries for scan line n+1 into the active list while processing scan line n. These new entries are easily placed directly into the proper position in the active list. With 2000 vectors spread over 512 scan lines there will only be about 4 additions per scan line on the average. However, this number could vary up to 500 additions for some unusual images.

A design that handles either lines or surfaces is only slightly more complex than either FIGURE 2 or 3, and would be the more reasonable implementation choice. It is simply the union of the main parts of both FIGURES 2 and 3 with a few switches located at points where the two diagrams differ.

Since the active list data is sorted on Xc there is no need for the 512 word scan line buffers as before. Instead a single word register contains the current beam intensity/color. The processing of the active list can be synchronized to the current X-value of the raster scan. As the raster sweeps across a scan line from left to right, X changes by one unit each 132ns. The current active list buffer is scanned in synchronism at this rate. When encountering a non-empty entry (i.e. the left side of some surface is encountered), data is loaded from the active list into a register. The I portion of the data in this register continually specifies the beam intensity/color. An adder (which easily works in 132ns) adds dx/dy to Xc, and the register contents are placed in a new active list buffer at location

Xc+ dx/dy (provided Ye ● current scan line y-value). This new active list buffer will be the input buffer for the next scan line.

The memory required for scan line processing of surfaces is two 2048 9-bit word active list buffers (we need not explicitly store the integer part of X for each active list entry). No 512 word line buffer is needed.

The cost (complexity) of such a special-purpose processor as estimated from a design using 7400 series logic, is approximately the same as for the processor described in section B. A comparable video generator is also needed as in B.

D. POSSIBLE MODIFICATIONS/ENHANCEMENTS TO THE PROCESS

If the scan line processor was implemented in software or used a slower inexpensive microprocessor that was not able to execute the algorithm within one frame time on some complex pictures, a modified design could be used so that the X position on each scan line where I changes value would be stored in an encoded manner in a buffer. The length of this buffer is proportional to the picture complexity, and is generally much smaller than a frame buffer memory. It could be used to keep the display refreshed for many frames by a relatively simple hardware device to generate the video signal. This, of course, precludes displaying a different image on each frame.

One advantage of partitioning the processing between the Y-sort and the scan line processors in the manner described above is that several hidden surface algorihms which produce raster scan output[6,7,8] use, at an intermediate step, an ACTIVE LIST containing the data sorted in the same order as the ACTIVE LIST described above. Thus it would be possible to replace the scan line processor described above with a more complex processor that would discover the overlapping surfaces from the ACTIVE LIST and produce a display with hidden surfaces removed. The Y-sort processor could still be used to produce the active list. In this case the I value in the ACTIVE LIST would normally contain an identification number for each surface. Also, an even number of entries would appear in the ACTIVE LIST for each surface, corresponding to the edges where the scan ray enters and exits the surface as it moves left to right along one horizontal scan line. The intensity of each different surface is supplied by indexing in an intensity table, using the surface identification number as index. If such a processor was not capable of processing the entire image in one frame time, the intensity change buffer just described could be used to buffer the image for several frames.

IV. COMPARISON OF FRAME-BUFFER AND PROCESSOR METHODS

Table 1 compares the performance limiting factors of the frame-buffer and processor approach. A frame buffer for 512x512 resolution requires 262,144 9-bit words of memory. The processor approach described in this paper requires only 30,000 9-bit words of memory (29,000 words for surfaces), or about 1/9 as much memory as the frame buffer method.

The frame buffer approach requires some processing power to generate the intensity patterns for vectors from their endpoint descriptions. To meet the 2000 vector per frame specification this requires a processor capable of processing 1 vector and storing the results in a frame buffer each 17us. In the processor approach, the scanline processor performs this function on the fly. The main difference between methods here is that the processor approach must do this calculation in real time, whereas the frame buffer approach may do it more slowly at the expense of real time motion.

The frame buffer approach has no component corresponding to the Y-sort processor. Therefore, the equipment tradeoff between the two methods is a Y-sort "fast microprocessor" and a scan line hardware processor vs. 232,000 9-bit words of memory and enough host processing power to generate the intensity patterns for vectors. Since the scan line processor performs essentially the same algorithm , we may, to a first approximation, equate the scan line processor to the cost of the host processing power needed to generate the intensity frame buffer patterns for individual vectors. An informal survey of the current literature shows that 232,000 words of memory has a cost many times that of a "fast microprocessor". We are considering only component costs here, supposing the fabrication costs for 232,000 words of memory is approximately equal to assembly costs of a "fast microprocessor". This assumption is based on today's approximately equal integrated circuit count for both the memory and processor described by FIGURE 3.

V. SUMMARY

The algorithm and suggested implementation using microprocessors is not proported to be the best such algorithm or implementation. However, it does show the capability of a microprocessor and a small special purpose processor to perform the raster scan conversion process. Thus the use of this technique appears feasible. Economically, we conclude that unless the ratio of processor to memory costs changes drastically from its current value, implementation without a frame buffer appears to be preferred, based on today's component costs, for systems wih high resolution or real time motion. Less readily comparable differences in the two raster conversion processes are the maximum picture complexity limits imposed by processor speeds vs. the maximum real-time motion picture complexity imposed by frame buffer memory speed and host bit-map generating speed. Also not readily comparable are the differences in software required by the loss of a frame buffer and the addition of a vector list describing the image. The frame buffer allows easy reading of the current image at a given (x,y) point. Lieberman[9] notes that this makes it easy to discover the edges of any enclosed region in the image, or to find one's way out of a maze. On the other hand, the existence of a vector list describing an image allows transformations to be performed easier. A frame buffer system by Garrett[10] even includes a vector list for this purpose. Note that in the approach described in this paper, real-time motion of the entire displayable image is automatic, provided the host computer or yet another dedicated microprocessor can generate the input data in real time. On the other hand, with a frame buffer, any arbitrary memory intensity/color pattern can be displayed (flicker free), even with moderately slow memory, so long as it does not all move in real time.

Performance parameter effected.	Limited in frame-buffer approach by:	Limited in processor approach by:
XY resolution	size of frame buffer	speed of scanline processor.
Complexity of still picture	No limit within resolution.	Speed of all processors
Complexity of real-time motion images.	Ability of host to generate coordinates in real time. Speed of host CPU to interpolate between vector endpoints.	Abiliy of host to generate coordinates in real time. Speed of scanline processor.
	Write speed of frame buffer memory to accept new image bit map.	Speed of Y-sort processor.

TABLE 1.
FRAME BUFFER - PROCESSOR APPROACH COMPARISON

REFERENCES

[1] Jordan, B.W., and R.C. Barrett, "A Scan Conversion Algorithm with Reduced Storage Requirements", Communications of the ACM 16,11 (November 1973), pp. 676-681.

[2] Erdahl, Alan C., "Displaying Computer Generated Half-Tone Pictures in Real Time", Computer Science Department, University of Utah, RADC-TR-69-250, 1972.

[3] Myers, A.J., "A Digital Video Information Storage and Retrieval System", Third Annual Conference on Computer Graphics, Interactive Techniques, and Image Processing, Philadelphia, Pennsylvania, July, 1976, pp. 45-50.

[4] DIGITEK, Inc., Box 5486, Raleigh, North Carolina, 27607.

[5] Schottky Bipolar LSI Microcomputer Set, INTEL Corp, 3065 Bowers Ave., Santa Clara, California.

[6] Bouknight, W.J., "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations", Communications of the ACM, (13,9), September, 1970, pp. 527-536.

[7] Romney, B.W., G.S. Watking, and D.C. Evans, "Real-Time Disply of Computer Generated Half-Tone Perspective Pictures", Proceedings of the IFIP Congress 1968, pp. 973-978.

[8] Hamlin, G.A., C.W. Gear, "Raster-Scan Hidden Surface Algorithm Techniques", Proceedings of the Fourth Annual Conference on Computer Graphics, Interactive Techniques, and Image Processing, San Jose, California, July 1977.

[9] Lieberman, Henry, "The TV Turtle: A Logo Graphics System for Raster Displays", Proc. ACM Symposium on Graphics Languages, Miami Beach, Florida, April 1976, pp. 66-72.

[10] Garrett, Michael T., "An Interpretive/Interactive Subroutine System for Raster Graphics", Proc. ACM Symposion on Graphics Languages, Miami Beach, Florida, April 1976, pp. 101-105.

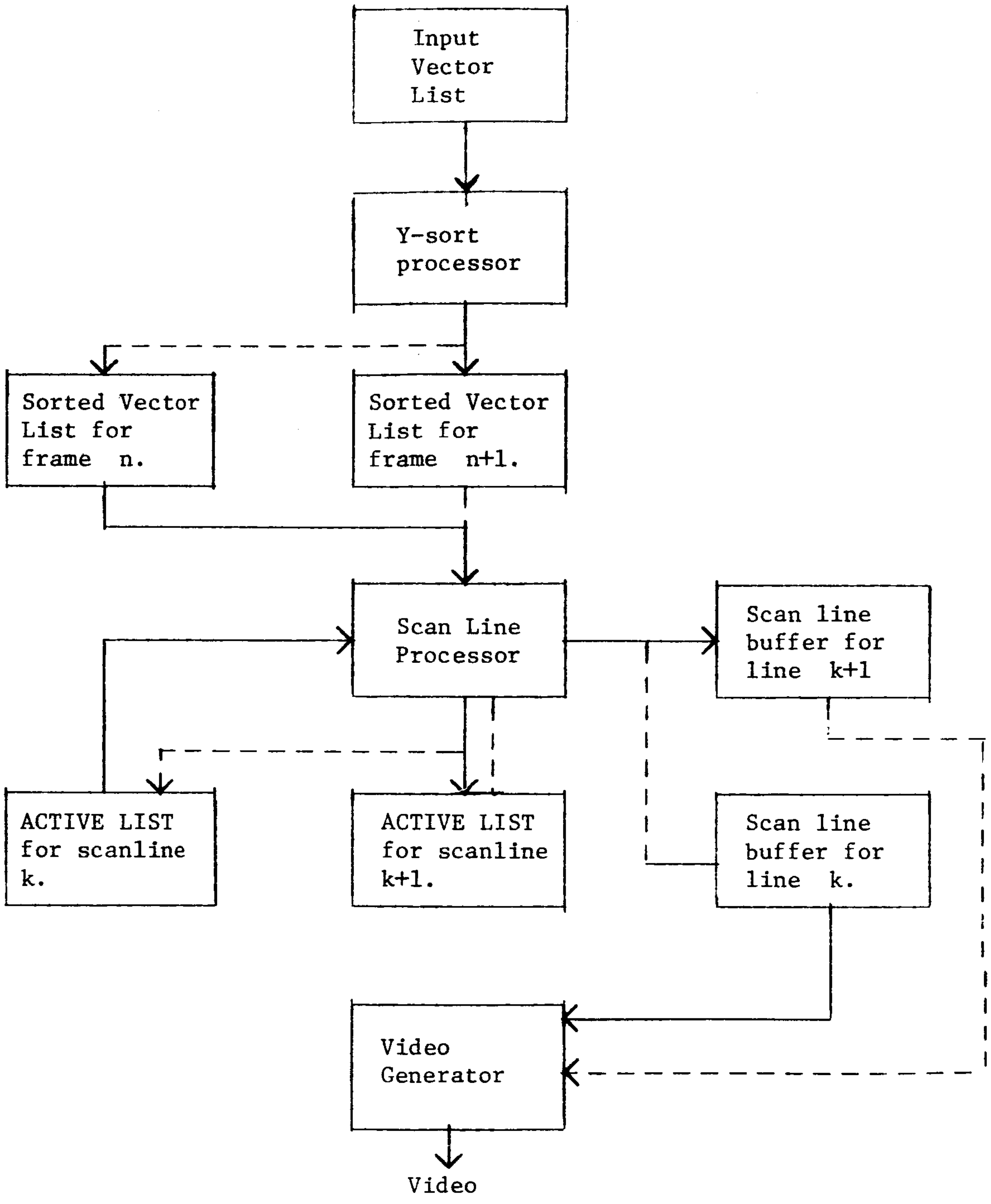
Input Vector List
Y-sort processor
Sorted Vector List for frame n.
Sorted Vector List for frame n+1.
Scan Line Processor
Scan line buffer for line k+1
ACTIVE LIST for scanline k.
ACTIVE LIST for scanline k+1.
Scan line buffer for line k.
Video Generator
Video

FIGURE 1

Scan Line Processor for Line Drawings

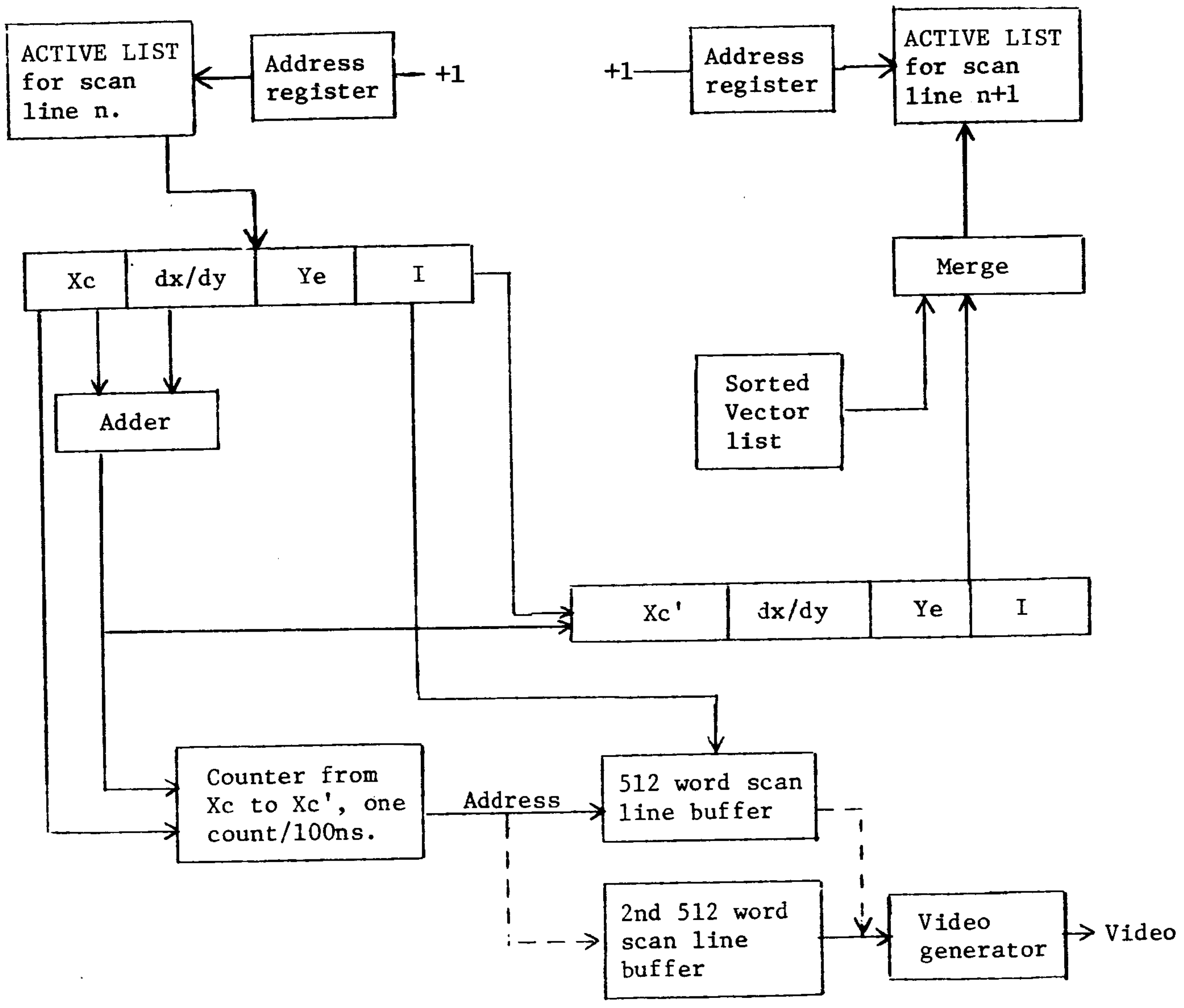
ACTIVE LIST for scan line n.
Address register
+1
+1
Address register
ACTIVE LIST for scan line n+1
Xc
dx/dy
Ye
I
Merge
Adder
Sorted Vector list
Xc'
dx/dy
Ye
I
Counter from Xc to Xc', one count/100ns.
Address
512 word scan line buffer
2nd 512 word scan line buffer
Video generator
Video

FIGURE 2

Scan Line Processor for Surface Drawings

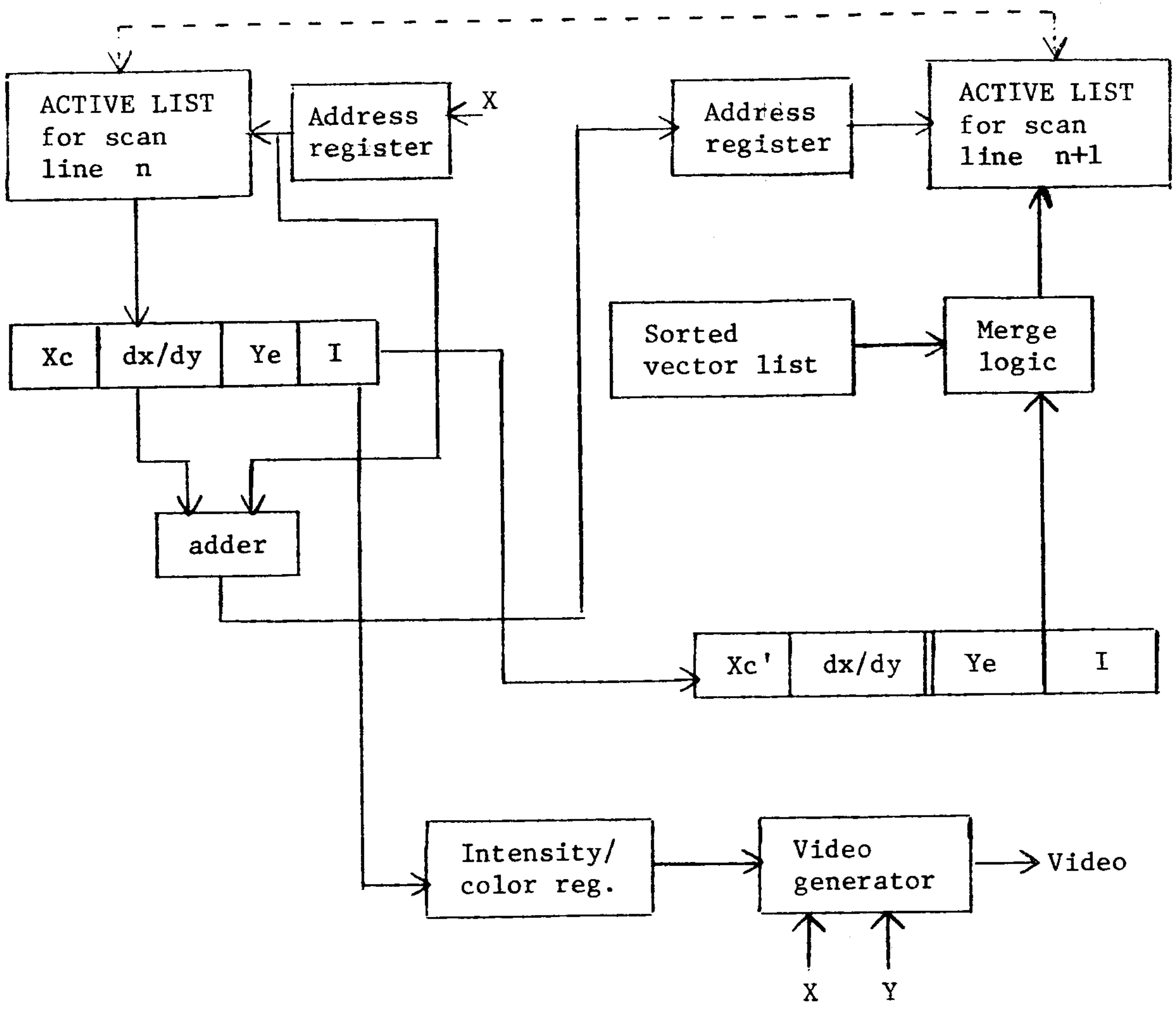

FIGURE 3